AN1163

USB HID Class on an Embedded Device

Author:	*Sean Justice*
	Microchip Technology Inc.

INTRODUCTION

The Human Interface Device (HID) is a class for use with Universal Serial Bus (USB). The HID class consists of devices that a human may use to control the operation of computer systems. Some of these devices include, but are not limited to: a mouse; a keyboard; a joystick; buttons; and switches. Along with providing information from human interfaces, there are provisions for various types of output to indicate action by the computer system.

This application note discusses and provides a HID device function driver that can be integrated with almost any application running on Microchip 32-bit PIC® microcontroller products with USB peripheral. The HID function driver uses the Microchip PIC32 USB device stack.

Along with the HID application note, a demo application is included that demonstrates the HID device function driver. This HID application demo simulates a mouse.

ASSUMPTIONS

The author assumes that the reader is familiar with the following Microchip development tools: MPLAB® IDE and MPLAB REAL ICE™ in-circuit emulator. It is also assumed that the reader is familiar with C programming language and USB device protocol and descriptors. Terminology from these technologies is used in this document and only brief overviews of the concepts are provided. Advanced users are encouraged to read the associated specifications.

FEATURES

This application note provides key components of a HID device class driver. The Microchip HID device class driver incorporates the following features:

• Functions independently of RTOS or application
• Supports Microchip MPLAB IDE tool suite
• Supports the HID 1.1 specification, as stated in "*Universal Serial Bus (USB) Device Class Definition for Human Devices (HID), Version 1.11*" (available on the Internet at the following URL: http://www.usb.org/developers/hidpage/)
• Uses only one USB endpoint
• Macros are provided to create HID Reports
• Handles multiple HID Reports
• Handles standard HID USB configuration requests, as stated in Chapter 9 of the *"Universal Serial Bus Specification, Revision 2.0"* (available at http://www.usb.org/developers/docs/)

LIMITATIONS

Since the HID is developed for use in embedded systems, the limitations are those that are inherited by the USB device stack (refer to Microchip Application Note AN1176, *"USB Device Stack for PIC32 Programmer's Guide"*).

SYSTEM HARDWARE

This application and firmware was developed for the following hardware:

• PIC32 Family Microcontroller PIM (Processor Interface Module), supporting USB
• Microchip Explorer 16 Development Board
• USB PICtail™ Plus Daughter Board

The USB device and HID function driver source files can be modified to use an alternative development board and accommodate most hardware differences.

AN1163

PIC® MCU MEMORY RESOURCE REQUIREMENTS

The HID function driver inherits all of the memory requirements of the USB device stack. Refer to the **"Memory Resource Requirements"** section of AN1176, *"USB Device Stack for PIC32 Programmer's Guide"* for more information.

The HID function driver consumes Flash and RAM memory as shown in the following table:

TABLE 1: MEMORY REQUIREMENTS

Memory	Size
Flash	3,284 Bytes
RAM	160 Bytes

The HID function driver application defines the following items:

• USB descriptor table
• HID report structure

The USB descriptor table and HID report structure, are required for any HID function driver application.

USB descriptor table memory requirements are shown in the following table:

TABLE 2: USB DESCRIPTOR TABLE

Memory	Size
Flash	56 Bytes

HID report structure memory requirements are shown in the following table:

TABLE 3: HID REPORT STRUCTURE

Memory	Size
Flash	1076 Bytes

The amount of memory resources consumed by the USB descriptor table and HID report may vary, depending on various factors, including, but not limited to, the following circumstances:

• Whether the user wishes to use multiple USB function drivers
  - Number of configurations, interfaces and endpoint configurations
• Size and definition of HID report

INSTALLING SOURCE FILES

The complete source for the Microchip HID function driver is available for download from the Microchip web site (**Appendix F: "Source Code for the HID Function Driver"**). The source code is distributed in a single Microsoft Windows® installation file.

Perform the following steps to complete the installation:

1. Execute the installation file. A Windows installation wizard will guide you through the installation process.

2. Before continuing with the installation process, you must accept the software license agreement by clicking **I accept**.

3. After completion of the installation process, you should see the following directory structure:

   a) `hid_device_driver` directory under `\PIC32 Solutions\Microchip\USB`. This directory contains the source files and documentation for the HID function driver.

   b) `usb_func_hid` directory under `PIC32 Solutions\ Microchip\Include\USB`. This directory contains the include files for the PIC32 USB device stack and HID function driver.

   c) `usb_hid_mouse_device_demo` directory under `\PIC32 Solutions`. This directory contains the demo project and source files for the HID function driver mouse demo.

4. Refer to the release notes for the latest version-specific features and limitations.

SOURCE FILE ORGANIZATION

The HID device class consists of multiple files that are organized in multiple directories. Table 4 shows the directory structure.

TABLE 4: HID SOURCE FILE DIRECTORY STRUCTURE

File	Directory	Description
`hid.c`	`\PIC32 Solutions\Microchip\USB\hid_device_driver`	USB HID device class driver
`hiddsc.tmpl`	`\PIC32 Solutions\Microchip\USB\hid_device_driver`	HID descriptor template
`hidreport.tmpl`	`\PIC32 Solutions\Microchip\USB\hid_device_driver`	HID report template
`usb_device_hid.h`	`\PIC32 Solutions\Microchip\Include\USB`	API defines and modifiable macros
`hidpri.h`	`\PIC32 Solutions\Microchip\USB\hid_device_driver`	Private function and macro defines
`hiddesc.h`	`\PIC32 Solutions\Microchip\Include\USB`	USB HID descriptors
`hidreport.h`	`\PIC32 Solutions\Microchip\Include\USB`	HID report defines

DEMO APPLICATION

An application that demonstrates the HID function driver by simulating a computer mouse is included with the Microchip HID function driver. This application is designed to run on the Explorer 16 development board with Microchip USB device stack software. However, the application can be modified to support any board.

The mouse-simulation demo application performs the following services:

• USB device enumeration for HID function driver
• Emulates a mouse when switch SW3 (RD6) on the Explorer 16 board is pressed – the following actions result in the specified movements:
  - Press switch SW6 (RD7) to move the mouse toward the right.
  - Press switches SW6 (RD7) and SW4 (RD13) to move the mouse toward the left.
  - Press switch SW5 (RA7) to move the mouse in an upward direction.
  - Press switches SW5 (RA7) and SW4 (RD13) to move the mouse in a downward direction.

AN1163

Programming the Demo Application

To program a target with the demo application, you must have access to an MPLAB REAL ICE in-circuit emulator. The following procedure assumes that you will be using MPLAB IDE . If not, please refer to your specific programmer's instructions.

1. Connect MPLAB REAL ICE in-circuit emulator to the Explorer 16 board or your target board.
2. Apply power to the target board.
3. Launch MPLAB IDE.
4. Select the PIC32 device supporting USB of your choice (required only if you are importing a hex file previously built).
5. Enable MPLAB REAL ICE as a programmer.
6. Import the previously build hex file into MPLAB, if you wish to use it.
7. If you are rebuilding the hex file, open the project file and follow the build procedure to create the application hex file.
8. The demo application contains necessary configuration options required for the Explorer 16 board. If you are programming another type of board, make sure that you select the appropriate oscillator mode from the MPLAB IDE configuration settings menu.
9. Select the "Programmer" menu option in MPLAB IDE, and click "Select Programmer->6 REAL ICE."
10. When MPLAB IDE has detected the REAL ICE in-circuit emulator and the PIC MCU, select the "Programmer" menu option and click "Program" to program the device.
11. After a few seconds, the message "Programming successful" is displayed. If it is not, check the board and MPLAB REAL ICE connections. Refer to MPLAB IDE and REAL ICE online help for further assistance.
12. Remove power from the board and disconnect the MPLAB REAL ICE cable from the target board.
13. Reapply power to the board and make sure that the LCD reads "PIC32 HID Device". If it does not, check your programming steps and repeat, if necessary.

Building the Demo Application

The demo application included in this application note can be built using the Microchip C32 C compiler. If required, port the source to the compiler that you customarily use with Microchip microcontroller products.

This application note includes a predefined mouse HID project file for use with MPLAB IDE. The project was created using a PIC32 device with USB. If a different device is used, the appropriate device must be selected through the MPLAB IDE menu command.

In addition, the demo application project uses additional include paths as defined in the "Build Options" of MPLAB IDE.

The following include paths are required:

- `.\`
- `..\Microchip\Include`
- `..\..\Microchip\Include`

Table 5 lists the source files that are necessary to build
the demo application.

TABLE 5:   DEMO APPLICATION PROJECT FILES

File	Directory	Description
mouse_demo.c	\PIC32 Solutions\usb_hid_mouse_device_demo	Main demo source file
mouse_dsc.c	\PIC32 Solutions\usb_hid_mouse_device_demo	Mouse HID USB descriptors
mouse_report.c	\PIC32 Solutions\usb_hid_mouse_device_demo	HID report for the mouse demo
HardwareProfile.h	\PIC32 Solutions\usb_hid_mouse_device_demo	Hardware defines for the PIC32
usb_config.h	\PIC32 Solutions\usb_hid_mouse_device_demo	USB specific defines for helper functions
hid.c	\PIC32 Solutions\Microchip\USB\hid_device_driver	USB HID source file
hidpri.h	\PIC32 Solutions\Microchip\USB\hid_device_driver	Private function and macro definitions
hid.h	\PIC32 Solutions\Microchip\Include\USB	USB HID include file
hiddesc.h	\PIC32 Solutions\Microchip\Include\USB	HID specific descriptor defines
hidreport.h	\PIC32 Solutions\Microchip\Include\USB	HID report structure macros and defines
usb_device.c	\PIC32 Solutions\Microchip\USB	USB device APIs
usb_hal.c	\PIC32 Solutions\Microchip\USB	USB hardware APIs
usb_hal_core.c	\PIC32 Solutions\Microchip\USB	USB hardware core APIs
usb.h	\PIC32 Solutions\Microchip\USB	USB top-level include file
usb_ch9.h	\PIC32 Solutions\Microchip\Include\USB	USB defines and support, as in Chapter 9 of the *"Universal Serial Bus Specification, Revision 2.0"*
usb_common.h	\PIC32 Solutions\Microchip\Include\USB	USB common defines
usb_device.h	\PIC32 Solutions\Microchip\Include\USB	USB device defines and API prototypes
usb_hal.h	\PIC32 Solutions\Microchip\Include\USB	USB hardware support
mstimer.c	\PIC32 Solutions\Microchip\Common	1 millisecond timer
ex16lcd.c	\PIC32 Solutions\Microchip\Common	Explorer 16 Development Board LCD
mstimer.h	\PIC32 Solutions\Microchip\Include	1 millisecond timer defines
ex16lcd.h	\PIC32 Solutions\Microchip\Include	Explorer 16 Development Board LCD defines

AN1163

The following is a high-level procedure for building the demo application. This procedure assumes that you are familiar with MPLAB IDE and will be using MPLAB IDE to build the application. If not, refer to the instructions for your programmer to create and build the project.

1. Make sure that source files for the Microchip HID function driver are installed. If not, please refer to the **"Installing Source Files"** section.
2. Launch MPLAB IDE and open the project file.
3. Use MPLAB IDE menu commands to build the project. Note that the demo project is created to compile properly when the source files are located in the wizard-recommended directory structure. If you have moved or installed the source files to another location, you must recreate or modify existing project settings to build. See **"Building the Demo Application"** for more information.
4. The build process should finish successfully. If not, make sure that your MPLAB IDE and compiler are setup correctly.

Application-Specific USB Support

In using the Microchip PIC32 USB device firmware stack, the HID demo implements the following application-specific tables:

- USB Descriptor Table
- Endpoint Configuration Table
- Function Driver Table

THE USB DESCRIPTOR TABLE

Every USB device must provide a set of descriptors (data structures) that describe the device and provide details to the USB host about which class drivers to use. These descriptors are provided, and the information they contain, are clearly defined in Chapter 9 of the *"Universal Serial Bus Specification, Revision 2.0"* and *"Universal Serial Bus (USB) Device Class Definition for Human Devices (HID), Version 1.11"*. Refer to these documents for complete details.

The USB device descriptors can be organized into three groups:

- Device
- Configuration
- Strings

The device descriptor identifies the type of device and gives the number of possible configurations.

The configuration descriptors describe the types of interfaces and endpoints used. This group also includes class-specific descriptors.

The string descriptors, although generally optional, provide user-readable information that the host may display.

Demo Application Descriptor Table

The descriptor table that is provided by the demo application is included in the source file `mouse_dsc.c` and is outlined in **Appendix E: "USB Descriptor Table Definitions"**.

The demo application descriptor table can be modified to add other interfaces or configurations. However, it is advisable that a thorough understanding of Chapter 9 of the *"Universal Serial Bus Specification, Revision 2.0"* and other applicable device function-driver-specific specifications is achieved before attempting to modify a descriptor table.

THE ENDPOINT CONFIGURATION TABLE

The endpoint configuration table is used by the USB device stack to properly configure all endpoints by interface and alternate setting as defined by the descriptor table. The table identifies which function driver will be used to service events that occur on each endpoint.

Each table entry contains the following information:

• Maximum packet size
• Configuration flags
• Configuration number
• Endpoint number
• Interface number
• Alternate setting
• Index in device function table for the endpoint handler

The endpoint configuration table for the demo application contains one entry because the HID function driver only requires a single endpoint (Interrupt-In). The following table is found in the source file `mouse_dsc.c`. Refer to AN1176, *"USB Device Stack for PIC32 Programmer's Guide"* for further information on the endpoint configuration table.

EXAMPLE 1:   ENDPOINT CONFIGURATION TABLE

```
const EP_CONFIG _EpConfigTlb[] =
{
    {
        HID_MAX_REPORT_SIZE,            // max pack size
    USB_EP_TRANSMIT|USB_EP_HANDSHAKE, // configure for Tx and enable
                                       // handshaking
    1,                                 // configuration number
    1,                                 // endpoint number
    0,                                 // interface number
    0,                                 // alternate setting
    0                                  // handler funciton index
    }
};
```

FUNCTION DRIVER TABLE

Since a device may implement more then one class or vendor-specific USB device function driver, the Microchip PIC32 USB device stack uses a table to manage access to support the function driver(s). Each table entry contains the information necessary to manage a single function driver.

Each table entry contains the following information:

• Initialization routine
• Event handler routine
• Initialization flags

The function driver table for the demo applications contains one entry because there is only one function driver, HID. The following table can be found in the source file `mouse_dsc.c`. Refer to Microchip Application Note AN1176, *"USB Device Stack for PIC32 Programmer's Guide"* for further information on the function driver table.

EXAMPLE 2:   FUNCTION DRIVER TABLE

```
const FUNC_DRV _DevFuncTbl[] =
{
    {   HIDInit,              // Initialization routine
        HIDEventHandler,      // Event handler routine
        0                     // Initialization flags
    }
};
```
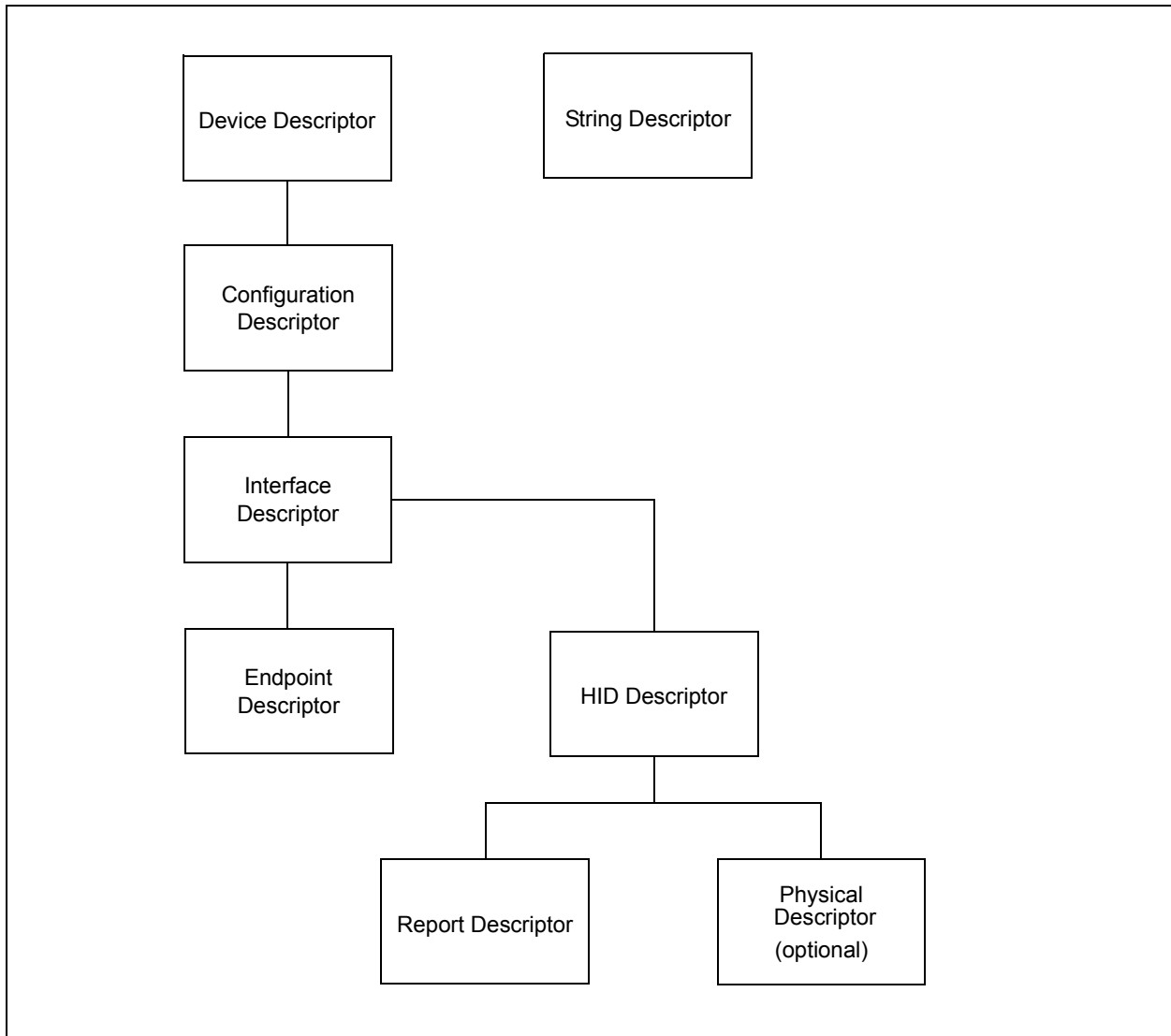
HID FUNCTION DRIVER OVERVIEW

HID Descriptor

Every USB device has descriptor structure associated with it. Each device may contain multiple classes, e.g., HID, which are defined in the interface layer. Figure 1 shows a descriptor structure tree that would describe an HID class device.

The HID descriptor indicates how many other HID-class-specific descriptors follow it. There must be at least one report descriptor. Physical descriptors are optional. A report descriptor describes the format and meaning of a data report generated by the device. The HID report descriptor is loaded by the host's HID class driver using a class-specific request. After being initialized, the device generates reports to indicate when a person interacts with it.

FIGURE 1:   USB DESCRIPTOR FOR HID FUNCTION DRIVER

AN1163

The HID function driver (USB device) communicates with the HID class driver (USB host) using default pipe (Control) or an Interrupt-In pipe. The Interrupt-In pipe is required for the HID function driver to transmit its data, but an optional Interrupt-Out endpoint can also be used, if needed.

Reports generated by the host to the device will be transmitted either through the Interrupt-Out endpoint or the default endpoint (Control) using the Set Report request.

TABLE 6: HID ENDPOINT CONFIGURATION

Pipe	Description	Required
Control (Endpoint 0)	USB control, class request codes, and polled data (Message data)	Y
Interrupt-In	Data-in, data from device	Y
Interrupt-Out	Data-out, data to the device	N

HID Report Descriptor

The HID report descriptor consists of pieces of information called items. Each item describes one aspect of the of the report data.

A report item follows a generic format of a one-byte prefix and then a payload. The prefix byte contains the tag, type, and size of the payload. Figure 2 shows the format of an item's prefix byte.

REPORT ITEM TYPES

Items are divided into the following types:

* Short Items
* Long Items

A short item has an optional data size that may be 0, 1, 2 or 4 bytes. Figure 3 shows an example of a short item payload size of 2 bytes.

A long item can have a size of up to 255 bytes. The format of the long type is the one-byte prefix, which is hard coded at 0xFE, one-byte size of the payload, a one-byte tag, and the payload. Figure 4 shows the format of a long item.
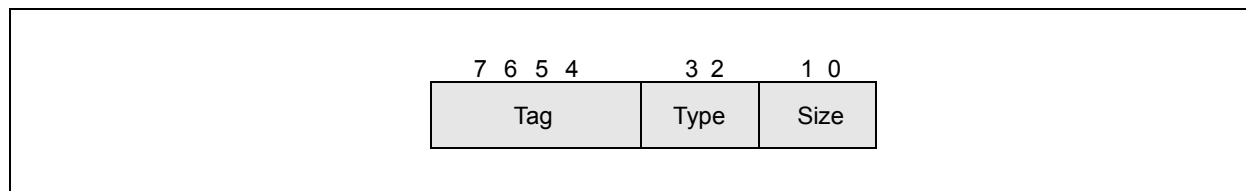
FIGURE 2: HID REPORT ITEM PREFIX BYTE

	7 6 5 4	3 2	1 0
	Tag	Type	Size

FIGURE 3: HID REPORT SHORT ITEM OF 2 BYTES

Tag	Type	Size	[data]	[data]
Bytes:	0		1	2

FIGURE 4: HID REPORT LONG ITEM FORMAT

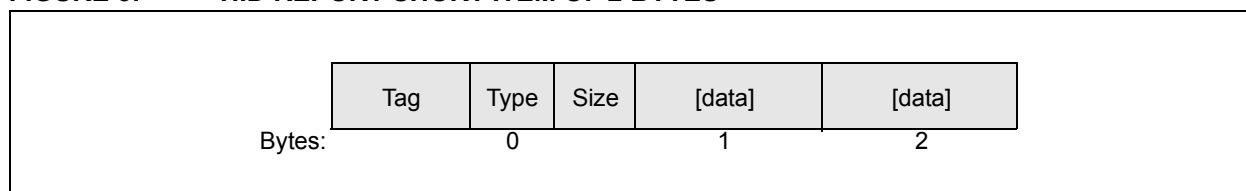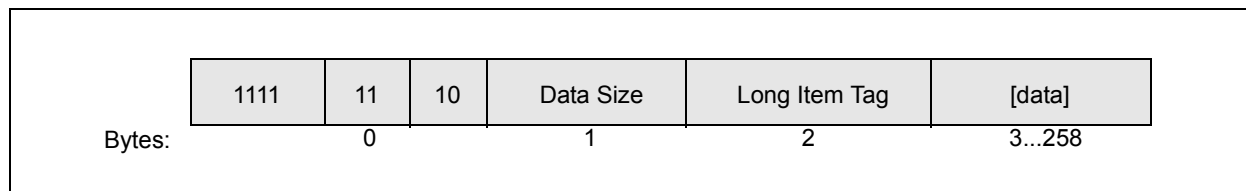| 1111	11	10	Data Size	Long Item Tag	[data]
Bytes:	0		1	2	3...258

Report Items

The items are divided into the following classes:

- Input Item
- Output Item
- Feature Items

Input Item class describes information about data provided by a physical control, like a keyboard or mouse.

Output Item class describes the data sent to the device, e.g., the LEDs on the keyboard that indicate whether the Caps Lock key is toggled on.

Feature Item class describes configuration information that can be sent to the device.

The following example presents a report descriptor that is included in the HID function driver demo:

```
Usage Page (Generic Desktop),
Usage (Mouse),
Collection (Application),
    Usage (Pointer),
    Collection (Physical),
        Usage Page (Button Page),
        Usage Minimum (1),
        Usage Maximum (3),
        Logical Minimum (0),
        Logical Maximum (1),
        Report Count (3),
        Report Size (1),
        Input (Data, Variable, Absolute),
        Report Count (1),
        Report Size (5),
        Input (Constant),
        Usage Page (Generic Desktop),
        Usage (X),
        Usage (Y),
        Logical Minimum (-127),
        Logical Maximum (127),
        Report Count (2),
        Report Size (8),
        Input (Data, Variable, Relative),
    End Collection,
End Collection
```

Each line represents a report item was used by the HID function driver to define the report structure in Figure 5.

- Report items **Usage Page (Generic Desktop)** and **Usage Page (Mouse)** tell the HID host that the report structure (about to be defined) is for a desktop and the device is a mouse.
- The data that is represented in the report is defined by the items defined in the **Collection (Physical)** section.

    The three buttons (B1-B3 of Figure 5) in the report are defined by:

    - **Usage Minimum (1)** and **Usage Maximum (3)** means that the mouse will use at least 1 button and a maximum of 3.
    - **Logical Minimum (0)** and **Logical Maximum (1)** is the range of values that the report will contain. In this case, the button will either be on **(1)** or off **(0)**.
    - **Report Count (3)** and **Report Size (1)** define the number and size of each button. The report contains 3 buttons which are represented by 1-bit (Byte 0 Bits 0-2 in the report).
    - **Input (Data, Variable, Absolute)** identifies what type of data is being sent. The buttons are input data to the host; they vary, and are absolute (on or off).

    The remaining bits of Byte 0 are filled with hard-coded zeros.
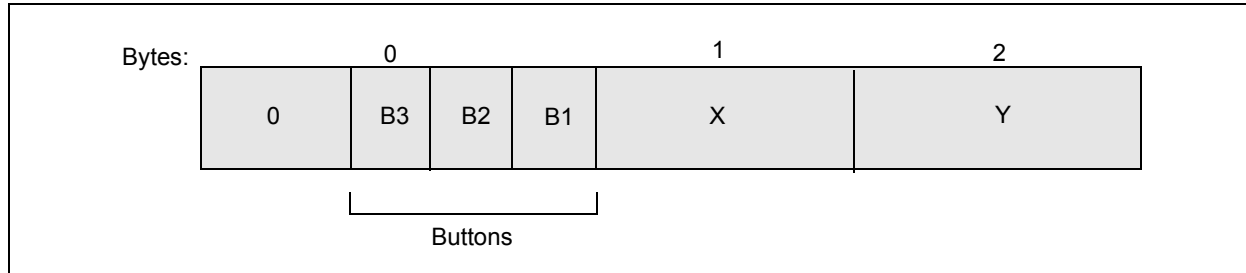
    - **Report Count (1)**
    - **Report Size (5)**
    - **Input (Constant)**

    The relative position of the mouse is represented by Byte 1 **(X)** and Byte 2 **(Y)**.

    - **Usage (X)** and **Usage (Y)** identify the next two inputs as a position.
    - **Logical Minimum (-127)** and **Logical Maximum (127)** are the range of value that the report will contain.
    - **Report Count (2)** and **Report Size (8)** define the number and size of each position.
    - **Input (Data, Variable, Relative)** identifies what type of data is being sent. The positions are input data to the host; they may vary, and report the relative position to the last report.

FIGURE 5:   MOUSE REPORT STRUCTURE

Reports

HID function drivers may send or receive a USB transaction every 1 millisecond. One or more USB transactions create a transfer of meaningful data to the device.

Table 7 shows the types of HID reports.

TABLE 7:   DATA TRANSFER REPORTS

Report Type	Pipe
Input	Interrupt-In
Output	Control or optional Interrupt-Out
Feature	Control or optional Interrupt-Out

HID reports may or may not have a report ID, one-byte prefix, associated with it. If a HID function driver can generate multiple reports, a report ID is required, tagging the report to insure proper data handling. If no report ID item tags are present in the report descriptor, it is assumed that only one Input, Output and Feature report structure exists. All data in reports are represented in little-endian format (see Figure 5).

Input reports are used to send data about user interactions with the device to the host. Output reports are used by the host to send control data to the device (such as a command to turn on the caps-lock LED). Feature reports are used by the host to configure the device. This demo only supports a single input report.

CONCLUSION

This document and the associated custom demo application consisting of a mouse demo using the HID function driver.

Normally, managing the Universal Serial Bus requires that a developer handle protocols for device identification, control, and data transfer. However, Microchip has taken care of the USB details and provided a simple HID function driver to make implementing applications simple for developers who use supported Microchip microcontrollers.

REFERENCES

- Microchip Application Note AN1176, *"USB Device Stack for PIC32 Programmer's Guide"*
- Microchip MPLAB® IDE
  In-circuit Development Environment, available free of charge, by license, from
  www.microchip.com/mplabide
- *"Universal Serial Bus Specification, Revision 2.0"*
  http://www.usb.org/developers/docs
- *"Universal Serial Bus (USB) Device Class Definition for Human Devices (HID), Version 1.11"*
  http://www.usb.org/developers/docs/HID1_11.pdf

AN1163

APPENDIX A: MICROCHIP HID FUNCTION DRIVER DEPENDENCIES

`HIDEventHandler` – USB Device Event Handler Function

The Microchip HID function driver application note provides an event handler that is compliant with Microchip's AN1176, *"USB Device Stack for PIC32 Programmer's Guide"* application note. This routine must be placed into the defined USB device function driver table.

Syntax

```
PUBLIC BOOL HIDEventHandler(USB_EVENT event, void *data, UINT size)
```

Parameters

`event` – Enumerated data type identifying the event that has occurred

`data` – A pointer to event-dependent data

`size` – The size of the event-dependent data, in bytes.

Return Values

`TRUE` if successful; `FALSE`, if not

Remarks

`HIDInit` – USB Device Initialization Handler Function

This routine initializes all data structures associated with the HID function driver. This routine must be placed into the defined USB device function driver table.

Syntax

```
PUBLIC BOOL HIDInit(unsigned long flags)
```

Parameters

`flags` – reserved, pass a 0

Return Values

`TRUE`, if initialization passed; else, `FALSE`

Remarks

The flags parameter is passed as the initialization flags parameter of the USB device function driver table.

Note:	Refer to Application Note AN1176, *"USB Device Stack for PIC32 Programmer's Guide"* for more information on `HIDEventHandler` and `HIDInit`.

APPENDIX B: HID FUNCTION REPORT HANDLER TABLE

To output reports from the HID class driver, the routines that are necessary to handle and get reports need to be supplied. The table, `_HidReportHandler`, is located in the HID source file (`hid.c`).

Each table entry contains the following information:

• Report Handler Routine
• Get Report Routine

API Definitions

REPORT HANDLER ROUTINE

This routine is called to handle Output reports.

Syntax

```
BOOL < Report Handler Routine >(void *data, unsigned int size)
```

Parameters

`data` – a pointer to the data that has been received from the HID class driver

`size` – the size of the passed data, in bytes

Return Values

If handled, `TRUE`; else `FALSE`

Precondition

Must be placed in the `_HidReportHandler` structure, in the HID source file

Side Effects

None

GET REPORT ROUTINE

This routine gets the current input report.

Syntax

```
UINT < Get Report Routine>(void *data)
```

Parameter

`data` – a pointer to the data buffer for the routine to place the report

Return Values

If `0, ERROR`; else, the size of the report in bytes

Precondition

Must be placed in the `_HidReportHandler` structure in the HID source file

Side Effects

None

APPENDIX C: HID FUNCTION DRIVER MACROS

The USB HID Class on an Embedded Device application note provides several function driver macros to customize it for an application.

The following HID function driver macros are available in the Microchip USB HID Class on an Embedded Device:

- `mHIDOpenTimer`
- `HID_TIMER_CONFIG`
- `mHIDConfigIntTimer`
- `HID_INT_CONFIG`
- `mHIDEnableTimerInt`
- `mHIDDisableTimerInt`
- `mHIDTimerIntHandler`
- `mHIDClearInt`
- `HID_EP_IN`
- `HID_EP_IN_SIZE`
- `HID_MAX_REPORT_SIZE`
- `HID_NUM_REPORTS`

mHIDOpenTimer

    Purpose: Macro to open a timer, using its peripheral library macro

    Default: `OpenTimer1`

HID_TIMER_CONFIG

    Purpose: Timer configuration value

    Default: `(T1_ON)`

mHIDConfigIntTimer

    Purpose: Macro to configure the timer interrupt, using its peripheral library macro

    Default: `ConfigIntTimer1`

HID_INT_CONFIG

    Purpose: Timer interrupt configuration value

    Default: `(T1_INT_OFF|T1_INT_PRIOR_2| T1_INT_SUB_PRIOR_1)`

mHIDEnableTimerInt

    Purpose: Macro to enable the timer interrupt, using its peripheral library macro

    Default: `EnableIntT1`

mHIDDisableTimerInt

    Purpose: Macro to disable the timer interrupt, using its peripheral library macro

    Default: `DisableIntT1`

mHIDTimerIntHandler

    Purpose: Macro to define the interrupt handler function

    Default: `Timer1IntHandler`

mHIDClearInt

    Purpose: Macro to clear the timer interrupt flag, using its peripheral library macro

    Default: `mT1ClearIntFlag`

HID_EP_IN

    Purpose: Endpoint from which the HID function driver will send input reports

    Default: 1

HID_EP_IN_SIZE

    Purpose: Size of the HID function driver endpoint, in bytes

    Default: 16

HID_MAX_REPORT_SIZE

    Purpose: Maximum size of a total report structure, in bytes

    Default: 64

HID_NUM_REPORTS

    Purpose: Number of HID reports that are supported

    Default: 1

APPENDIX D: USB HID FUNCTION DRIVER API

This section describes the HID function driver API.

API - HIDSendReport

This routine sends an input report to the HID class driver.

Syntax

```
BOOL HidSendReport(BYTE id, void *data, UINT size, BOOL change)
```

Parameters

id – report ID; if no report ID is being used, the value is 0

data – pointer to the report data being sent

size – size of the data being sent

change – if the data has changed

Return Values

TRUE, if data send request has been serviced; else FALSE

Precondition

USB device stack and HID have been initialized

Side Effects

None

Example

```
// send a report
BYTE report[4];
report[0] = 'M';
report[1] = 'C';
report[2] = 'H';
report[3] = 'P';
if(!HidSendReport(0, report, 4, FALSE)
{
    // handle send error
}
```

AN1163

APPENDIX E: USB DESCRIPTOR TABLE DEFINITIONS

The HID function driver defines its descriptor table, as shown in the **"HID Descriptor"** section, with the values shown in the following tables:

TABLE E-1: DEVICE DESCRIPTOR

Field	Description	HID Function Driver Value
bLength	Size of this descriptor	12h
bDescriptorType	Type, always USB device descriptor	1
bcdUSB	USB specification version in BCD	200h
bDeviceClass	Device class code	0
bDeviceSubClass	Device sub-class code	0
bDeviceProtocol	Device protocol	0
bMaxPacketSize	Endpoint 0 max packet size	10h
idVendor	Vendor ID (VID)	4D8h
idProduct	Product ID (PID	Dh
bcdDevice	Device release number in BCD	1
iManufacturer	Manufacturer name string index	1
iProduct	Product description string index	2
iSerialNum	Product serial number string index	0
bNumConfigurations	Number of supported configurations	1

TABLE E-2: CONFIGURATION DESCRIPTOR

Field	Description	HID Function Driver Value
bLength	Size of this descriptor	9
bDescriptorType	Type, always USB configuration descriptor	2
wTotalLength	Total length of all descriptors	22h
bNumInterfaces	Number of interfaces	1
bConfiguration/Value	ID value	1
iConfiguration	Index of the string descriptor	0
bmAttributes		
reserved_zero	Always 0	0
remote_waking	1 if device supports remote wake-up	1
self_powered	1 if device is self-powered	0
reserved_one	Always 1	1
bMaxPower	milliamps/2 (e.g., 100 ma = 50)	50

TABLE E-3:   INTERFACE DESCRIPTOR

Field	Description	HID Function Driver Value
bLength	Size of this descriptor	9
bDescriptorType	Type, always USB interface descriptor	4
bInterfaceNumber	ID number of the interface	0
bAlternateSetting	ID number of the alternate interface setting	0
bNumEndpoints	Number of endpoints in this interface	1
bInterfaceClass	USB interface class ID	3
bInterfaceSubClass	USB interface sub-class ID	1
bInterfaceProtocol	USB interface protocol ID	2
iInterface	Interface description sting index	0

TABLE E-4:   HID INTERFACE DESCRIPTOR

Field	Description	HID Function Driver Value
bLength	Size of this descriptor	9
bDescriptorType	Type, always USB HID interface descriptor	21h
bcdHID	HID specification release in BCD	101h
bCountryCode	Country code of the localized hardware	0
bNumDescriptors	Number of class descriptors, minimum of 1	1

TABLE E-5:   HID CLASS INTERFACE DESCRIPTOR

Field	Description	HID Function Driver Value
bDescriptorType	ID type of the class descriptor	22h
bDescriptorLength	Total size of the descriptor (i.e., HID report)	32h

TABLE E-6:   ENDPOINT DESCRIPTOR

Field	Description	HID Function Driver Value
bLength	Size of this descriptor	7
bDescriptorType	Type, always USB endpoint descriptor	5
bEndpointAddress		
ep_num	Endpoint number	1
reserved	Always 0	0
direction	IN or OUT	1
bmAttribute		
transfer_type	Transfer type	1
synch_type	Synch type	0
usage_type	Usage type	0
reserved	Always 0	0
wMaxPacketSize	Largest packet the endpoint can handle	16
bInterval	Time between polling this endpoint for data	10

TABLE E-7:   SERIAL STRING DESCRIPTOR

Field	Description	HID Function Driver Value
bLength	Size of this descriptor	4
bDescriptorType	Type, always USB string descriptor	3
wLangid[]	String	409h

TABLE E-8:   MANUFACTURE STRING DESCRIPTOR

Field	Description	HID Function Driver Value
bLength	Size of this descriptor	34h
bDescriptorType	Type, always USB string descriptor	3
wLangid[]	String	Microchip Technology Inc.

TABLE E-9:   PRODUCT STRING DESCRIPTOR

Field	Description	HID Function Driver Value
bLength	Size of this descriptor	2Eh
bDescriptorType	Type, always USB string descriptor	3
wLangid[]	String	Mouse In a Circle Demo

APPENDIX F: SOURCE CODE FOR THE HID FUNCTION DRIVER

The complete source code for the Microchip USB HID Class on an Embedded Device driver is offered under a no-cost license agreement. It is available for download as a single archive file from the Microchip corporate web site, at:

www.microchip.com.

After downloading the archive, always check the release notes for the current revision level and a history of changes to the software.

AN1163

REVISION HISTORY

Rev. A Document (02/2008)

This is the initial released version of this document.

Note the following details of the code protection feature on Microchip devices:

• Microchip products meet the specification contained in their particular Microchip Data Sheet.

• Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.

• There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.

• Microchip is willing to work with the customer who is concerned about the integrity of their code.

• Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rfPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, PICkit, PICDEM, PICDEM.net, PICtail, PIC^{32} logo, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2008, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

♻ Printed on recycled paper.

QUALITY MANAGEMENT SYSTEM

CERTIFIED BY DNV

═ ISO/TS 16949:2002 ═

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

01/02/08